24. C Toolkit Resources for C++ Toolkit Users

Created: April 1, 2003
Updated: September 16, 2003

Summary

For certain tasks it becomes necessary to use, or at least refer to, material from the NCBI C Toolkit. This page simply collects a variety of links relevant to making use of the C Toolkit in the C++ Toolkit environment.

- Working with the NCBI C Toolkit
- C Toolkit Documentation
- C Toolkit Queryable Source Browser

Using NCBI C and C++ Toolkits together

Note: Due to security issues, not all links in the public version of this file could be accessible by outside NCBI users. Unrestricted version of this document is available to inside NCBI users at: http://intranet.ncbi.nlm.nih.gov/ieb/ToolBox/CPP_DOC/libs/c_cxx.html.

- Overview
- Shared Sources
    - CONNECT Library
    - ASN.1 Specifications
- Run-Time Resources
    - LOG and CNcbiDiag
    - REG and CNcbiRegistry
    - MT_LOCK and CRWLock
    - CONNECT Library in C++ Code
        - Setting LOG
        - Setting REG

- Setting MT-Locking
- Convenience call CONNECT_Init()

- C Toolkit diagnostics redirection
- CONNECT Library in C Code
  - Convenience call CONNECT_Init()

Overview

When using both C and C++ Toolkits together in a single application, it is very important to understand that there are some resources shared between the two. This document describes how to safely use both Toolkits together, and how to gain from their cooperation.

Shared Sources

To maintain a sort of uniformity and ease in source code maintenance, `CONNECT` library is the first library of both Toolkits kept same at the source code level. To provide data interoperability, ASN.1 specifications have to be identical in both Toolkits, too.

CONNECT Library

`CONNECT` library is at the moment the only part of C code of both Toolkits, which is kept same in its entirety in both Toolkits. The old API of `CONNECT` library is still supported by means of simple wrapper code, which is situated in the C Toolkit only. There is an external safety script, which periodically (controlled by a *cron* daemon) checks and maintains source file indentity. Conventionally, all development for `CONNECT` library is done within the C++ Toolkit tree. When a modified source file is committed to the CVS repository but is not yet updated in the C Toolkit tree, the safety script detects the discrepancy, and then could be used to eliminate it by copying the newer file over. If for some reason the modified version is mistakenly checked into C Toolkit tree then the safety script alerts the situation.

ASN.1 Specifications

On a contrary to `CONNECT` library, the ASN.1 data specifications are maintained within C Toolkit source structure, and have to be copied over to C++ Toolkit tree whenever they are changed. There is the same (as for `CONNECT` library) safety script, which keeps "an eye" on those changes, and sends an alert when C++ Toolkit ASN.1 specs go out of sync with their C Toolkit counterparts.

The full set of tools, which maintain identity of both `CONNECT` library and ASN.1 specifications can be found in directory *scripts/internal/c_toolkit*.

However, the internal representations of ASN.1-based objects differ between the two toolkits. If you need to convert an object from one representation to the other, you can use the template class ***CAsnConverter<>***, defined in *ctools/asn_converter.hpp*.

Run-Time Resources

Being written for use "as is" in the NCBI C Toolkit and yet to be in the C++ Toolkit tree, `CONNECT` library could not employ directly all the utility objects offered by the C++ Toolkit such as message logging ***CNcbiDiag***, registry ***CNcbiRegistry***, and MT-locks ***CRWLock***. All these objects were replaced with helper objects coded entirely in C (as tables of function pointers and data).

On the other hand, throughout the code `CONNECT` library refers to predefined objects `g_CORE_Log` (so called *CORE C logger*) `g_CORE_Registry` (*CORE C registry*), and `g_CORE_Lock` (*CORE C MT-lock*), which actually are never initialized by the library, i.e. they are empty objects, which do nothing. It is an application's resposibility to replace these dummies with real working logger, registry, and MT-lock. There are two approaches, one for C and another one for C++ application.

In a C program *connect/ncbi_util.h* with calls to ***CORE_SetREG()***, ***CORE_SetLOG()***, and ***CORE_SetLOCK()*** can be used to set up the registry, the logger, and the MT-lock, correspondingly. There are even more convenience routines concerning *CORE logger*, like ***CORE_SetLOGFILE()***, ***CORE_SetLOGFILE_NAME()***, which facilitate redirecting logging messages to either a C stream (***FILE****) or a named file.

In a C++ program, yet another additional step is necessary of converting *native* C++ objects, by calls declared in *connect/ncbi_core_cxx.hpp* and as described later in this section, into their C equivalents, so that the C++ objects could be used where types ***LOG***, ***REG*** or ***MT_LOCK*** are expected.

LOG and *CNcbiDiag*

`CONNECT` library has its own logger, which has to be set by any of routines declared in *connect/ncbi_util.h*: ***CORE_SetLOG()***, ***CORE_SetLOGFILE()*** etc. On the other hand, the interface defined in *connect/ncbi_core_cxx.hpp* provides the following C++ function to convert a logging stream of the NCBI C++ Toolkit into a ***LOG*** object:

```
LOG LOG_cxx2c (void)
```

which creates the ***LOG*** object on top of the corresponding C++ ***CNcbiDiag*** object, and then both C and C++ objects could be manipulated interchangeably, causing exactly the same effect on the underlying logger. Then, the returned C handle ***LOG*** can be subsequently used as a *CORE C logger* by means of ***CORE_SetLOG()***, like in the following nested calls: *CORE_SetLOG (LOG_cxx2c());*

REG and CNcbiRegistry

connect/ncbi_core_cxx.hpp declares the following C++ function to bind C ***REG*** object to ***CNcbiRegistry*** used in C++ programs built with the use of the NCBI C++ Toolkit:

```
REG REG_cxx2c (CNcbiRegistry* reg, bool pass_ownership = false)
```

Similarly to *CORE C logger* setting, the returned handle can be later used with ***CORE_SetREG()*** declared in *connect/ncbi_util.h* to set up the global registry object (*CORE C registry*).

MT_LOCK and CRWLock

There is a function

```
MT_LOCK MT_LOCK_cxx2c (CRWLock* lock, bool pass_ownership = false)
```

declared in *connect/ncbi_core_cxx.hpp*, which converts an object of class ***CRWLock*** into a C object ***MT_LOCK***. The latter can be used as an argument to ***CORE_SetLOCK()*** for setting the global *CORE C MT-lock*, used by a low level code, written in C. Note that passing 0 as the lock pointer will effectively create a new internal ***CRWLock*** object, which will then be converted into ***MT_LOCK*** and returned. This object gets automatically destroyed when the corresponding ***MT_LOCK*** is asked to do so. If the pointer to ***CRWLock*** is passed non `NULL` then the second argument can specify whether resulting ***MT_LOCK*** acquires the ownership of the lock, thus is able to delete the lock when destructing itself.

CONNECT Library in C++ Code

Setting LOG

To set up the *CORE C logger* to use the same logging format of messages and destination as used by ***CNcbiDiag***, the following sequence of calls may be used:

```
CORE_SetLOG(LOG_cxx2c());
SetDiagTrace(eDT_Enable);
SetDiagPostLevel(eDiag_Info);
SetDiagPostFlag(eDPF_All);
```

Setting REG

To set the *CORE C registry* be the same as C++ registry ***CNcbiRegistry***, the following call is necessary:

```
CORE_SetREG(REG_cxx2c(cxxreg, true));
```

here `cxxreg` is a ***CNcbiRegistry*** registry object created and maintained by a C++ application.

Setting MT-Locking

To set up a CORE lock, which is used throughout the low level code, including places of calls of non-reentrant library calls (if no reentrant counterparts were detected during configure process), one can place the following statement close to the beginning of the program:

```
CORE_SetLOCK(MT_LOCK_cxx2c());
```

Note that the use of this call is extremely important in a multi-threaded environment.

Convenience call CONNECT_Init()

Header file *connect/ncbi_core_cxx.hpp* provides convenience call, which sets all shared `CONNECT`-related resources discussed above for an application program written within the C++ Toolkit framework (or linked solely against the libraries contained in the toolkit):

```
void CONNECT_Init(CNcbiRegistry* reg = NULL);
```

The call takes only one argument, an optional pointer to a registry, which is used by the application, and should also be considered by the CONNECT library. No registry will be used if NULL gets passed. The ownership of the registry is passed along. This fact should be noted by an application doing extensive use of CONNECT stuff in static classes, i.e. prior to or after main(), because the registry can get deleted before CONNECT library stops using it. The call also ties *CORE C logger* to ***CNcbiDiag***, and privately creates *CORE C MT-lock* object (on top of ***CRWLock***) for internal synchronization inside the library.

An example on how to use this call could be found in the test program *test_ncbi_conn_stream.cpp*. It shows how to properly setup *CORE C logger*, *CORE C registry* and *CORE C MT-lock* in order for them to use the same data both in C and C++ parts of both the library and the remaining code of the application.

Another good source of information is working application examples found in *src/app/id1_fetch*.

Note from the examples that the convenience routine does not change logging levels or disable/enable certain logging properties. If this is desired, the application still has to use separate calls.

C Toolkit diagnostics redirection

In a C/C++ program linked against both NCBI C++ and NCBI C Toolkits the diagnostics messages (if any) generated by either Toolkit are not necessarily directed through same route, which may result in lost or garbled messages. To set the diagnostics destination be the same as ***CNcbiDiag***'s one, and thus to guarantee that the messages from both Toolkits will be all stored sequentially and in the order they were generated, there is a call

```
#include <ctools/ctools.h>
void SetupCToolkitErrPost(void);
```

which is put in a specially designated directory *ctools* providing back links to the C Toolkit from the C++ Toolkit.

CONNECT Library in C Code

CONNECT library in C Toolkit has a header *connect/ncbi_core_c.h*;, which serves exactly the same purposes that does *connect/ncbi_core_cxx.hpp* described previously. It defines an API to convert native Toolkit objects, like logger, registry, and MT-lock into their abstract equivalents, ***LOG***, ***REG***, and ***MT_LOCK***, respectively, which defined in *connect/ncbi_core.h*, and subsequently can used by CONNECT library as *CORE C* objects.

Briefly, the calls are:

- ```
  LOG LOG_c2c (void);
  ```

  Create a logger ***LOG*** with all messages sent to it rerouted via the error logging facility used by the C Toolkit.
  ```

- ```
  REG REG_c2c (const char* conf_file);
  ```

  Build a registry object ***REG*** from a named file `conf_file`. Passing `NULL` as an argument causes the default Toolkit registry file to be searched for and used.

- ```
  MT_LOCK MT_LOCK_c2c (TNlmRWlock lock, int/*bool*/ pass_ownership);
  ```

  Build an ***MT_LOCK*** object on top of ***TNlmRWlock*** handle. Note that passing `NULL` effectively creates an internal handle, which is used as an underlying object. Ownership of the original handle can be passed to the resulting ***MT_LOCK*** by setting the second argument to a non-zero value. The internally created handle always has its ownership passed along.

Exactly the same way as described in previous section, all objects, resulting from the above functions, can be used to set up *CORE C logger*, *CORE C registry*, and *CORE MT-lock* of `CONNECT` library using the API defined in *connect/ncbi_util.h*: ***CORE_SetLOG()***, ***CORE_SetREG()***, and ***CORE_SetLOCK()***, respectively.

Convenience call CONNECT_Init()

As an alternative to using per-object settings shown in the previous paragraph, the following "all-in-one" call is provided:

```
void CONNECT_Init (const char* conf_file);
```

This sets *CORE C logger* to go via Toolkit default logging facility, causes *CORE C registry* to be loaded from the named file (or from the Toolkit's default file if `conf_file` passed `NULL`), and creates *CORE C MT-lock* on top of internally created ***TNlmRWlock*** handle, the ownership of which is passed to the ***MT_LOCK***.

Note again that properties of logging facility is not affected by this call, i.e. the selection of what gets logged, how, and where, should be controlled by using native C Toolkit's mechanisms defined in *ncbierr.h*.